© 2013 by Abhishek Sharma. All rights reserved.

HIGH THROUGHPUT PATH SELECTION FOR UNSTRUCTURED DATA CENTER NETWORKS

BY

ABHISHEK SHARMA

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2013

Urbana, Illinois

Advisor:

Assistant Professor P.Brighten Godfrey

Abstract

The increase in demand and popularity of cloud and big data applications has driven the need for higher throughput data center network design. Recent work to provide topologies with much denser interconnects pose a difficult challenge for routing of traffic within a data center. Even with proposals like MPTCP that improve upon TCP, there is a still a gap between theoretical throughput and empirical throughput.

Our goal is to study routing in data centers at both flow and packet-level to determine the cause of inefficiency and study methods of path selection that may help bridge the gap between optimal throughput and packet level throughput. The difference in throughput can be attributed to inefficiency due to path selection and inefficiency due to protocol overhead; we quantify the contribution of each. Focusing on path selection, our experiments show that *k-disjoint paths* provide much better throughput in most topologies than the previously used *k-shortest paths*. We also show that one can positively impact network throughput by varying the number of paths according to network density.

Table of Contents

Chapter 1 Introduction **1**

Chapter 2 Related Work **3**

Chapter 3 Simulation Setup **5**
3.1 Graph construction 5
3.2 Path selection 5
3.3 Traffic matrices 6
3.4 Throughput computation 6

Chapter 4 Results **7**
4.1 Quantifying the difference between optimal and packet-level throughput 7
4.2 k-disjoint paths vs. k-shortest paths at flow-level 9
4.3 k-disjoint paths vs. k-shortest paths at packet-level 11
4.4 Choosing number of paths 12

Chapter 5 Conclusion/Future Work **15**

References **16**

Chapter 1

Introduction

Cloud and big data applications are distributed across many thousands of machines. Transfering data between end to end systems can put a huge burden on the network and lead to congestion and hot spots. The network often becomes the bottleneck in large distributed applications. To tackle this issue, we require higher throughput network designs. We need network designs that let us transfer data with improved throughput, failure resistance and congestion avoidance.

Recent work in network design has led to a number of data centre topologies which aim to deal with the above mentioned issues by providing denser interconnects than the existing ones. However, this can lead to some interesting problems when routing traffic within a data centre. Raiciu et al [10] try to supplement the new data center network designs with the use of Multipath TCP (MPTCP), a TCP protocol based on multiple paths. The authors use *8-shortest paths* for path selection, thus creating 8 subflows per MPTCP connection. However, even though MPTCP with *8-shortest paths* provides better performance than TCP, it still lags behind optimal flow throughput as shown by Singla et al [12].

In this paper, we study and quantify the factors that lead to the difference in performance between optimal flow-level throughput and packet-level throughput. Selection of better paths for routing can help us bridge this throughput gap. For our experiments, we study and compare disjoint paths against shortest paths in Jellyfish [12], an unstructured data center topology. We also calculate network throughputs by using *k-shortest paths* for various k's in Jellyfish and by comparing those throughputs at both flow and packet level, suggest how many paths must be selected to obtain higher performance network routing.

We compare packet-level throughput obtained by using *8-shortest paths* against optimal flow-level throughput and attribute the difference in performance to suboptimal path selection and protocol overhead, with a substantial contribution from each. We further proceed to quantify the role of each to throw some insight on how much throughput improvement can be attained by focusing on better path selection. We observe from our results that, unless the network is either very dense or

very sparse, *k-disjoint paths* outperforms *k-shortest paths*. While sparse networks can lead to the selection of suboptimal disjoint paths, very dense networks render both disjoint paths and shortest paths as practically the same. We further show that network density also plays a role in deciding the number of paths that should be selected for routing. While protocol limitations might never let us achieve maximum throughput, by choosing better paths we can quantify the improvement that can be possibly made.

We should also mention that all our experiments are performed on Jellyfish and perfoming these experiments on other data centre topologies is part of future work.

Chapter 2

Related Work

The nature and scale of internet traffic has led to several research proposals for a next generation transmission protocol design which aims to substantially improve on the current protocol i.e. TCP. Specifically, there is a need to to improve congestion avoidance and also better utilize network resources. MPTCP [13], a TCP protocol based on multiple path data transmission is one such proposal. By utilizing multiple paths to transmit data in a connection between two network entities, network resources are utilized better and with better route selection data packets are distributed among suitable routes to reduce congestion. [10] and [11] look into the utilization of MPTCP within datecentres to improve networking and performance. The authors compared the performance of MPTCP vs single-path TCP in different data centre topologies and traffic patterns and found that MPTCP provides significant advantages in almost all of them. According to their discovery, it would take around eight subflows per MPTCP connection to properly($\sim$ 90 %) utilize the network (in FatTree [6], VL2 [2] and BCube [3]).

Singla et al proposed a new data centre network design, Jellyfish, based on random regular graphs to improve bandwidth and flexibility in data centres. They also advocate the use of MPTCP for congestion control by performing experiments that show that using *8-shortest* paths with MPTCP allows supporting a larger number of flows at higher throughput than both ECMP with TCP and *8-shortest paths* with TCP. They compared *k-shortest* path routing at the packet-level against optimal flow-level performance showing than Jellyfish's packet level throughput is approximately *86 %* of optimal throughput.

Our work attempts to build on the work of the Jellyfish authors by utilizing disjoint paths in a data centre using Jellyfish toplogy. We attempt to bridge the gap between optimal flow-level throughput and Jellyfish's packet-level throughput by suggesting the use of disjoint paths and trying to figure out the reason behind the gap in throughputs. We compare disjoint paths and *k-disjoint paths* with other shortest path related strategies. As an extension, we also check the performance of disjoint paths in simulated networks with both routing and congestion control involved using the

MPTCP simulator.

Two key factors in selecting multiple paths are how the paths are selected and how many paths are selected. We want to select good quality, short paths that minimize routing overhead and link failure chances. We also want to select only a limited number of paths because it is more compex to distribute traffic among a larger set of paths and the overhead associated with establishing, mainting and tearing down paths also increases. Past work on multipath selection has focused mainly on two significant areas: quick failure recovery and load-balancing and effective selection of multiple paths.

[9] and [15] propose an on demand multipath routing algorithm to better distribute load between routes and reduce packet loss. [5] further ventures into selecting two maximally disjoint paths and using routes on a per-packet basis.

[8] examines selection of paths based on reliability in a MANET. The authors propose selecting a set of edge-disjoint paths that is highly reliable and augmenting the set only if a candidate path increases the reliability of the set. [7] proposes a meta-heuristic approach to solve the routing selection problem.

Specifically, using disjoint paths has been an intriguing approach in multipath routing as they help in increasing path diversity, reducing path congestion and increasing fault tolerance among other things. Suurballe [14] and Bhandari [1] studied and proposed solutions to the problem of finding *k-disjoint paths* between two nodes in a graph. In general, selection of paths should provide a good balance between complexity and performance.

Chapter 3

Simulation Setup

3.1 Graph construction

The experiments conducted in this paper are all conducted on the Jellyfish topology. The topology is essentially a random regular graph constructed at the top-of-rack (ToR) switch layer. Each switch consists of p ports, it uses k ports to connect to servers and p - k ports to connect to other switches. The simplest case is to have the number of servers be a multiple of the number of switches, thus every switch has the same number of ports and servers. As k out of p ports are used to connect to servers, the graph constructed at the switch layer has a degree of p - k.

3.2 Path selection

To send flow between a source switch and a destination switch in the graph, we select a set of paths based on various path selection strategies. For each source-destination pair, depending upon the strategy in use, we figure out if flow should be sent through a link or not. Flow is sent through a link if and only if it is a part of the path(s) through which flow must be sent. For example, when sending flow from switch x to switch y using only shortest paths, flow is sent through an arbitrary link a - b if

path length of x - a + path length of b - y + 1 = shortest path length of x - y

The shortest path length can be pre-computed once the random regular graph and traffic matrix has been constructed.

We tested and compared the throughput obtained from the following route seclection strategies: no restrictions on path selection at all to get ideal flow-level performance, all paths of length less than equal to *shortest path + 4*, all paths of length equal to shortest path, *k-shortest paths* where *k = 4, 6, 8, 10, 12, 14, 16*, all disjoint paths and *8-disjoint paths*. It was found that all paths of length greater than equal to *shortest path + 4* gives us the approximately $\geq 99.5\%$ of ideal flow-level

performance and in some large size graphs where computing ideal flow-level performance was not possible because of the limitations of the machine involved the latter was used as the benchmark against which all are strategies were compared.

We would like to clarify that *k-shortest paths* refers to the first k shortest paths. For a source-destination pair, this may include paths whose lengths are longer than the shortest path length between the pair. Also, *k-disjoint paths* refers to the set of k edge-disjoint paths between the source and destination whose combined length is a minimum. We compute the set of *k-disjoint paths* using Bhandari's algorithm [1]. In cases where less than k disjoint paths exist between the source and destination, we pick the maximum possible number of disjoint paths that exist between the pair.

3.3 Traffic matrices

The traffic matrices used are random permutation traffic i.e. each source server randomly selects a destination server to send traffic to and conversely receives from a single other server. As multiple servers are mapped to single switch, at the switch layer graph a source switch can send flow to various destination switches. We should note that path selection has nothing to do with traffic patterns and as part of our future work, we would like to experiment on other traffic patterns apart from random permutation traffic.

3.4 Throughput computation

In our experiments, we have two methods to compute network throughput:

- using optimal fluid flow (subject to the constraints of topology and path selection), which we refer to as flow-level throughput. Flow is treated as splittable and fluid and the throughput is obtained by solving a multi-commodity network flow problem on the traffic matrix. We use the CPLEX linear programming solver [4] at the flow-level to solve this problem.

- using a packet level simulation of an actual transport protocol. We use the MPTCP simulator developed by the MPTCP authors.

Chapter 4

Results

4.1 Quantifying the difference between optimal and packet-level throughput

In this section, we compare the network throughput obtained by using *8-shortest paths* at both flow-level and packet-level against optimal flow-level throughput. The difference between optimal throughput and packet level throughput can be attributed to path selection and protocol overhead. Through our experiments we quantify how much each of them individually contributes to the inefficiency at the packet level.

We construct a graph of 100 switches and 800 servers (8 servers/switch) at the switch layer. In this case, the network degree is always *p - 8* where p is the total number of ports available to a switch. We vary the network degree by changing the number of ports available for each switch. *Figure 4.1* illustrates the performance of *8-shortest paths* in both the flow and packet-level simulator. We observe that as the network degree (i.e. number of ports) increases the performance of *8-shortest paths* detoriates when compared to optimal flow-level performance. This is because while the total number of paths increases as we increase the network degree, we continue to select only the 8-shortest options neglecting a large number of paths.

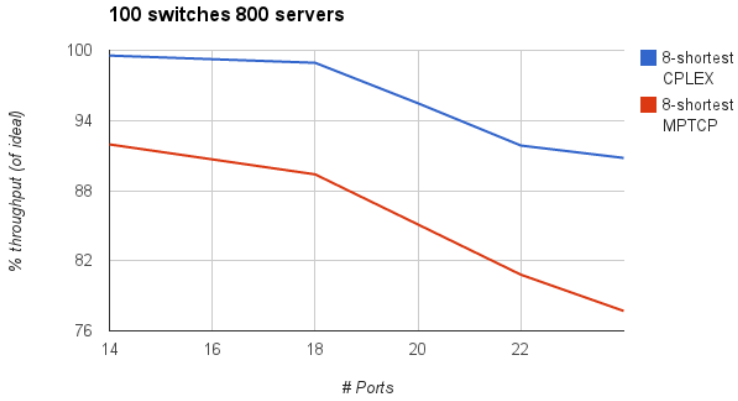

Figure 4.1: Comparing optimal flow-level throughput with both 8-shortest paths at both flow and packet-level.

Table 4.1 categorizes the difference in optimal throughput and packet-level throughput into inefficiency caused by path selection and inefficiency caused by protocol overhead. As we infer from the data, as the graph density increases the inefficiency caused by path selection can increase to approximately 40 - 45 %. Currently, since flows in the MPTCP simulator are restricted to values of 1, getting the true throughput of graphs with larger than 24 total ports is not possible. But by allowing flows of value larger than 1, one can test what happens with ports greater than 24.

# Switches	# Servers	# Ports	Inefficiency due to path selection(%)	Inefficiency due to protocol overhead(%)
100	800	14	5.56	94.43
100	800	18	10.04	89.95
100	800	20	30.39	69.60
100	800	22	42.43	57.56
100	800	24	41.28	58.71

Table 4.1: Breaking down inefficiency into path selection and protocol overhead

4.2 k-disjoint paths vs. k-shortest paths at flow-level

The authors of MPTCP and Jellyfish used *k-shortest paths* to obtain multiple paths to connect each source-destination switch pair. In this section, we compare the use of *k-shortest paths* and *k-disjoint paths*. We exclusively use *8-disjoint paths* and *8-shortest paths* to maintain consistency with the experiments performed by the authors of Jellyfish, who compared *8-shortest paths* packet-level throughput with optimal flow-level throughput. Our results lead us to conclude that in almost all networks, regardless of size and density, *8-disjoint paths* gives better throughput than *8-shortest paths*.

Figure 4.2 shows the results of throughput obtained by using both *8-disjoint paths* and *8-shortest paths* in a network with 100 switches and 400 servers. We consider this as our baseline.

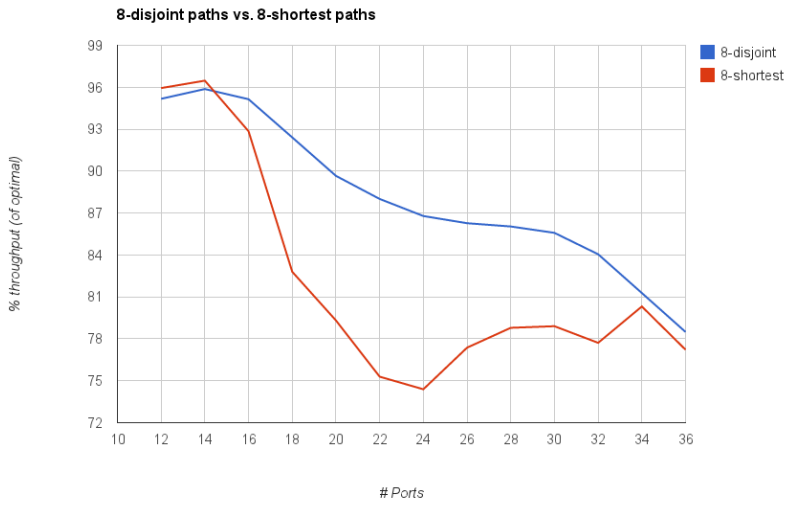

Figure 4.2: Comparing 8-disjoint paths vs. 8-shortest paths at flow-level for 100 switches 400 servers.

Figure 4.3 shows the results *8-disjoint paths* and *8-shortest paths* in a network with 100 switches and 800 servers. Here, we modify the number of servers per switch.

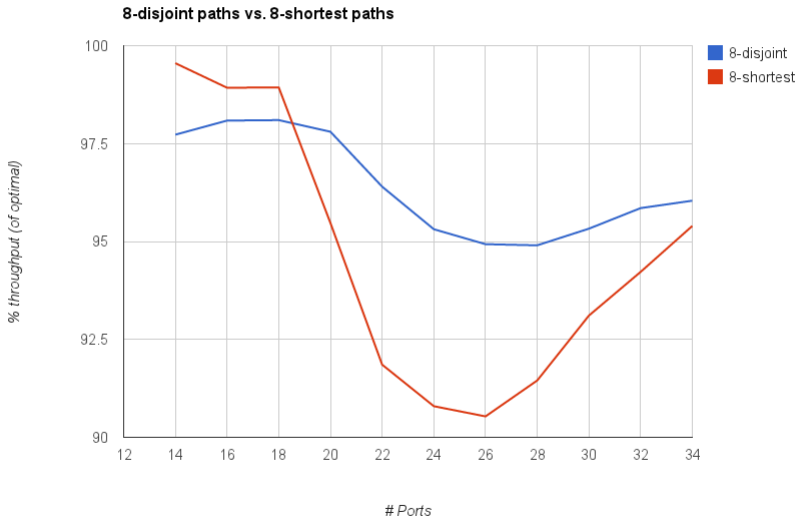

Figure 4.3: Comparing 8-disjoint paths vs. 8-shortest paths at flow-level for 100 switches 800 servers.

Figure 4.4 shows the results *8-disjoint paths* and *8-shortest paths* in a network with 150 switches and 600 servers. Here, we modify the number of switches while keeping the number of servers per switch constant.

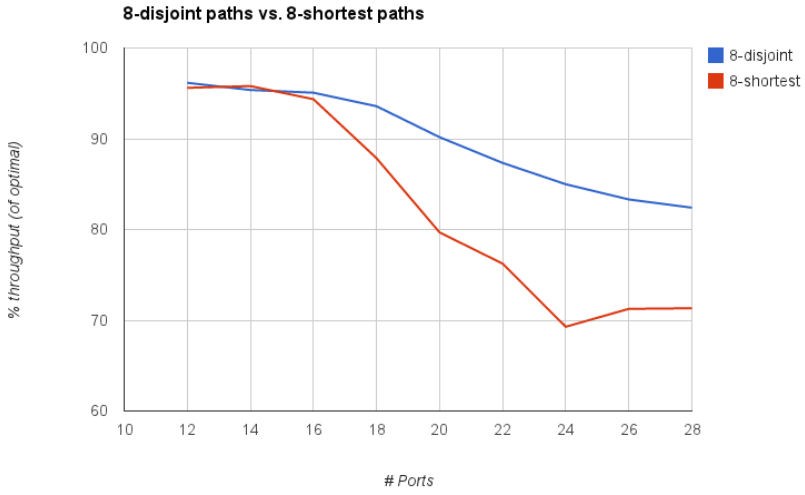

Figure 4.4: Comparing 8-disjoint paths vs. 8-shortest paths at flow-level for 150 switches 600 servers.

We notice a similar pattern from all three figures i.e. initially *8-shortest paths* tends to perform

better than *8-disjoint paths* but gradually as the network density increases throughput obtained from the latter performs significantly better than throughput obtained from the former. This trend carries on till a point where the difference between the throughputs obtained by both the strategies reaches its highest point before beginning to converge again.

The reason for this can be penciled to network density. In a fairly sparse network, with the network degree being $\leq 10\%$ of the total number of nodes in the network, there exists only d_n, the degree of the network, number of disjoint paths. Some of these paths are much longer than the paths chosen by shortest paths since one can repeatedly form shortest paths by modifying only one link. Choosing longer paths will have an adverse effect on the throughput.

However, as the density of the network increases the connectivity improves too, giving disjoint paths of comparable length to shortest paths. Morover, since disjoint paths consists of unique links only, they perform better than shortest paths in which a link might be repeated multiple times. Choosing disjoint paths in a graph with all links of the same capacity will lead to more flow.

As the network density continues to increase, after a point, the gap in performance between *8-shortest paths* and *8-disjoint paths* begins to reduce. This is because the increasing network density allows us to have an increasing number of disjoint paths that also double up as shortest paths i.e. the disjoint paths are infact the shortest paths too. So, no link repetition is necessary to obtain the *k-shortest paths*.

4.3 k-disjoint paths vs. k-shortest paths at packet-level

We further compared throughputs obtained from *8-disjoint paths* and *8-shortest paths* at the packet-level. Similar to our results from previous section, we observe that *8-disjoint paths* gives higher throughput than *8-shortest paths*.

As mentioned before, since the MPTCP simulator takes only flow values < 1, it only made sense to display a network with 100 switches and 800 servers with total number of ports limited to 24. *Figure 4.5* shows the network throughputs of *8-disjoint paths* and *8-shortest paths* as compared to optimal throughput



Figure 4.5: Comparing 8-disjoint paths vs. 8-shortest paths at packet-level for 100 switches 800 servers.

4.4 Choosing number of paths

[10] shows that using at most 8 subflows per MPTCP connection can achieve $\sim 90\%$ throughput in FatTree, VL2 and B-Cube. They argue that for more than 8 subflows, the increase in overhead to maintain the extra subflows offsets the improvement in network utilization. In this section, we compare network throughput obtained from *k-shortest paths* (and hence k subflows) for various k's ($k = 4, 6, 8, 10, 12, 14, 16$) at the flow-level. From our results, we see that as the value of k increases the rise in throughput begins to diminish. $k = 8$ appears to be a reasonable choice since it achieves the most gain in throughput. However, the gain in throughput, as we increase k, varies according to the degree of the network.

We use the same numbers for graph sizes as in *section 4.2* to get the following three graphs:

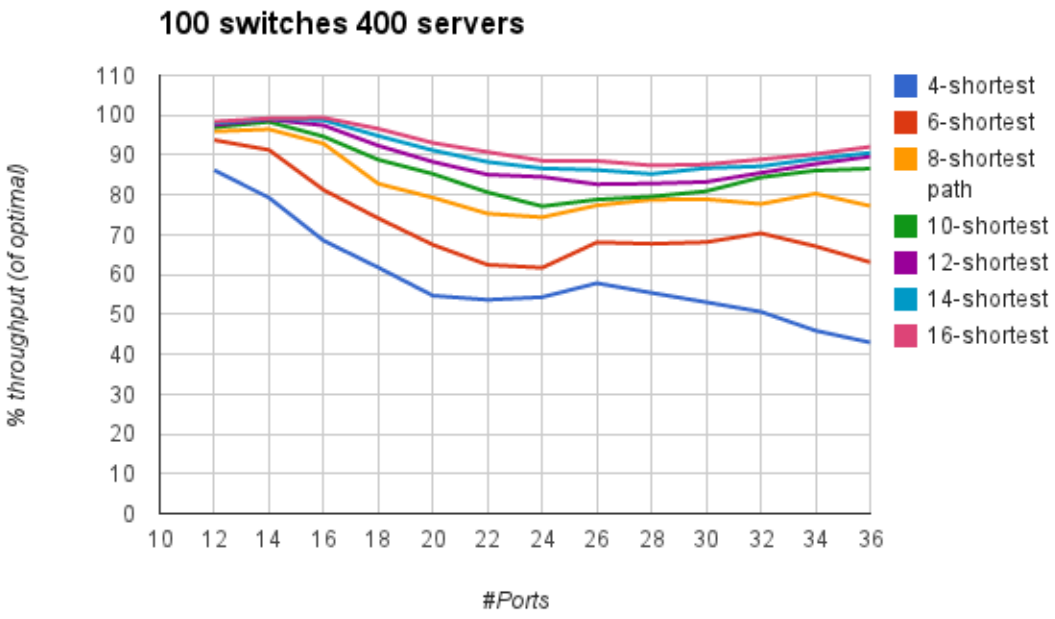

Figure 4.6: Comparing k-shortest paths at flow-level for 100 switches 400 servers.

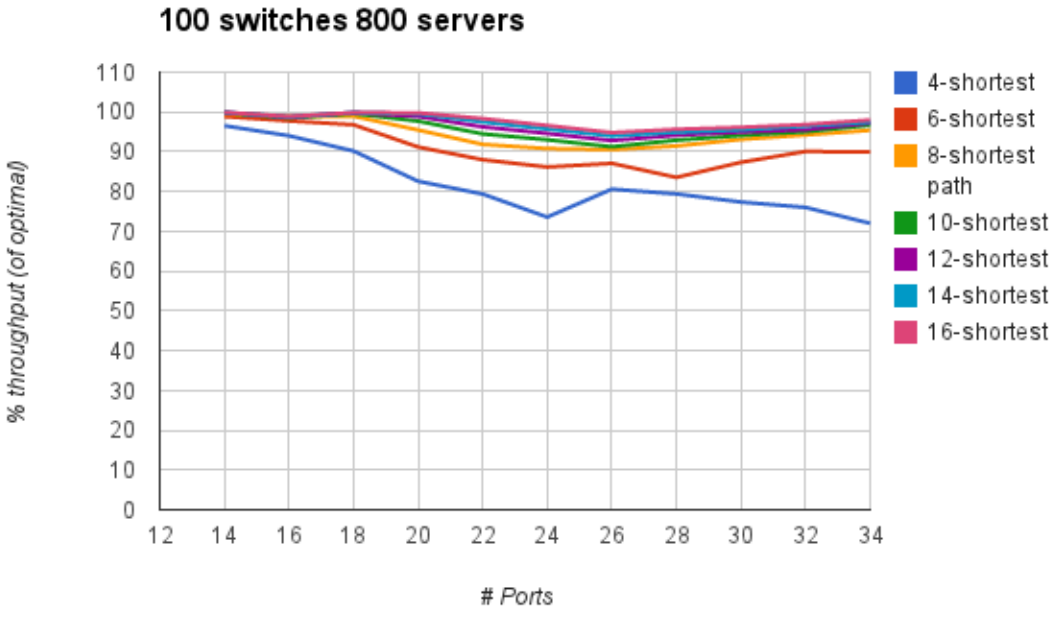

Figure 4.7: Comparing k-shortest paths at flow-level for 100 switches 800 servers.

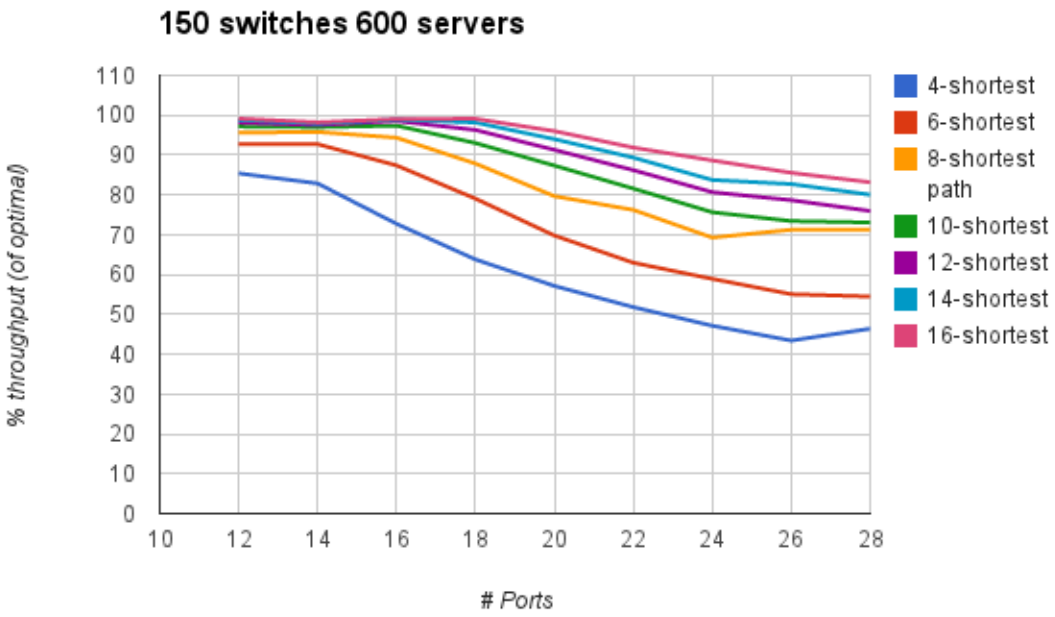

Figure 4.8: Comparing k-shortest paths at flow-level for 150 switches 600 servers.

From the three graphs we see that, as we gradually increase the value of k beyond 8, the improvement in throughput begins to diminish. However, for each of the three graphs, there is a range of network densities in which *16-shortest paths* does significantly better than the recommended *8-shortest paths*. We conclude that in Jellyfish to maintain 90% of the throughput, one would have to choose the number of subflows according to the density of the network. We hope to study the trends of the *k-shortest paths* for different k's in further detail as part of our future work.

Chapter 5

Conclusion/Future Work

Our simulations indicate that by selecting disjoint paths over shortest paths and a larger number of paths, we can reduce the effect that path selection has on the decrease in network throughput. We also show the effect network density plays in selecting both the type of paths (disjoint vs shortest) and the number of paths.

All our experiments were conducted on the Jellyfish topology. As part of our future work, we would like to conduct our experiments on other network interconnects (BCube, VL2) and traffic matrices.

References

[1] Ramesh Bhandari. *Survivable networks: algorithms for diverse routing.* Springer, 1999.

[2] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. *SIGCOMM Comput. Commun. Rev.*, August 2009.

[3] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: a high performance, server-centric network architecture for modular data centers. *SIGCOMM Comput. Commun. Rev.*, August 2009.

[4] Inc ILOG. Ilog cplex: High-performance software for mathematical programming and optimization, 2006.

[5] S.J. Lee and M. Gerla. Split multipath routing with maximally disjoint paths in ad hoc networks. In *IEEE International Conference on Communications, 2001. ICC 2001*, volume 10, 2001.

[6] Charles E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput.*, October 1985.

[7] Shiwen Mao, Y.T. Hou, Xiaolin Cheng, H.D. Sherali, and S.F. Midkiff. Multipath routing for multiple description video in wireless ad hoc networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, 2005.

[8] Panagiotis Papadimitratos, Zygmunt J. Haas, and Emin G Sirer. Path set selection in mobile ad hoc networks. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, MobiHoc '02, 2002.

[9] Marc R. Pearlman, Zygmunt J. Haas, Peter Sholander, Zygmunt J. Haas Peter Shol, and Siamak S. Tabrizi. On the impact of alternate path routing for load balancing in mobile ad hoc networks. In *Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing*, 2000.

[10] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving datacenter performance and robustness with multipath tcp. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, 2011.

[11] Costin Raiciu, Christopher Pluntke, Sebastien Barre, Adam Greenhalgh, Damon Wischik, and Mark Handley. Data center networking with multipath tcp. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, 2010.

[12] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. Jellyfish: networking data centers randomly. NSDI'12, 2012.

[13] Ulrich Sorger, Foued Melakessou, and Zdzislaw Suchanecki. Mptcp: Concept of a flow control protocol based on multiple paths for the next generation internet 2007. In *7th International Symposium on Communications and Information Technologies ISCIT'07*, volume Library of Congress: 2007929336. 2007.

[14] J.W. Suurballe. *Disjoint paths in a network*. 1974.

[15] A. Tsirigos and Z. J. Haas. Multipath routing in the presence of frequent topological changes. *Comm. Mag.*, 39(11), November 2001.